Multi-Application Inter-Tile Synchronization on Ultra-High-Resolution Display Walls

Sungwon Nam[1], Sachin Deshpande[2], Venkatram Vishwanath[3], Byungil Jeong[4], Luc Renambot[1], Jason Leigh[1]

[1] Electronic Visualization Laboratory
842 W. Taylor St.
Chicago, IL 60607
snam5, luc, spiff@evl.uic.edu

[2] Sharp Laboratories of America
5750 NW Pacific Rim Blvd.
Camas, WA 98607
sdeshpande@sharplabs.com

[3] Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439
venkatv@mcs.anl.gov

[4] Texas Advanced Computing Center
10100 Burnet Rd Bldg 196
Austin TX 78758
bijeong@tacc.utexas.edu

ABSTRACT

Ultra-high-resolution tiled display walls are typically driven by a cluster of computers. Each computer may drive one or more displays. Synchronization between the computers is necessary to ensure that animated imagery displayed on the wall appears seamless. Most tiled display middleware systems are designed around the assumption that only a single application instance is running in the tiled display at a time, therefore synchronization can be achieved with a simple solution such as a networked barrier. When a tiled display has to support multiple applications at the same time, the simple networked barrier approach does not scale. In this paper we propose and experimentally validate two synchronization algorithms to achieve low-latency inter-tile synchronization for multiple applications with independently varying frame-rates. The Two-Phase algorithm is more generally applicable to various high-resolution tiled display systems. The One-Phase algorithm provides superior results but requires the support for NTP, and is more CPU-intensive.

Categories and Subject Descriptor

I.3.2 [**Computer Graphics**]: Graphics Systems – *Distributed/network graphics*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *Client/server*; D.4.1 [**Operating Systems**]: Process Management – *Synchronization*

Keywords

Frame synchronization, Tiled-display, Cluster computing

1. Introduction

Ultra-high-resolution display walls are fast becoming a standard tool for scientific research. These types of displays are the only means by which scientists can see the massive data generated from their instruments and supercomputer simulations. With the advent of low-cost LCDs, researchers are now using tiled display walls as "mash up" environments where they can juxtapose a variety of data so that they can look at them as a whole [16]. While similar to the notion of Project Rooms or War Rooms of the past, a key difference is that for large-scale and collaborative scientific research, there is no other way to look at this magnitude of data. These projects routinely deal with time-varying data on the order of terabytes to petabytes. It is impossible to manage this information by printing out static diagrams on sheets of paper and pinning them to a wall as has been the traditional approach. The microscopes and telescopes used by scientists today are no longer simple optical instruments, but are integrated with complex computing systems that perform noise filtering, tiling, and feature detection. Ultra-high-resolution displays are becoming the new lenses that bring the data from these instruments into focus. To meet this challenge, the Electronic Visualization Laboratory at University of Illinois at Chicago has been conducting research with Sharp Labs of America on scalable display wall hardware and software. The culmination of this work is LambdaVision, a 100-Megapixel LCD wall, and the Scalable Adaptive Graphics Environment (SAGE) [16], a middleware system for driving such walls. Figure 1 shows the LambdaVision driven by SAGE that is used for the weekly meeting at Electronic Visualization Laboratory.

LambdaVision is an array of 55 LCD panels that are driven by a cluster of 28 computers. The computers cooperate to give the users the illusion of a seamless display environment. Therefore, precise coordination and synchronization between the computers are necessary to ensure that animated images that are displayed on the wall appear seamless. The Scalable Adaptive Graphics Environment (SAGE) has been developed for this purpose [31]. Unlike other tiled display middleware such as Chromium [15] or Equalizer [11], SAGE is designed at the outset to manage multiple images or animations from different applications at the same time enabling users to simultaneously access, stream, and juxtapose them on ultra-high-resolution tiled display walls.

The need to support multiple applications at the same time poses a significant challenge for image synchronization. Traditional

frame synchronization mechanisms used in systems such as Chromium or Equalizer do not scale due to increased message complexity when it is used in a system that is designed to support multiple applications simultaneously. In this paper, we propose and validate two new algorithms: a Two-Phase and a One-Phase frame synchronization algorithm to achieve low-latency inter-tile synchronization for multiple applications with varying frame-rates. The two algorithms achieve the same goal but differ in resource utilization and complexity. A key contribution of this paper is proposing a scalable way to achieve frame synchronization among display nodes in a tiled display wall that can support multiple applications simultaneously.

Figure 1. The SAGE in action. A student presenting at the weekly meeting in Electronic Visualization Laboratory.

In the sections below, we will discuss in greater detail, the synchronization requirements of ultra-high-resolution environments and related work. Next, we present background information about the SAGE. We will also detail the limitations of applying traditional synchronization approaches to SAGE and how our new algorithms provide significant improvements.

2. Related Work

The traditional model for driving tiled display walls is to use the entire surface to display a single visualization in exquisite resolution. However, as display walls began to grow in size and resolution, users found it more useful to be able to use the expansive screen estate and resolution for not simply displaying a single visualization, but multiples simultaneously so that they can compare them side by side [5, 9, 30, 32]. Middleware systems that fall under the former category include WireGL [14], Chromium [15], DMX [3], Equalizer [11], and CGLX [1]. In WireGL and its successor Chromium, one or more servers convert data from unmodified applications into OpenGL graphics primitives, which are then streamed to clients driving the tiled-display wall. CGLX does not distribute graphics primitives but runs the same copies of the OpenGL-based application on all clients and replicates the data on all the clients. Equalizer offers a hybrid approach where the user can combine various rendering techniques. DMX (Distributed Multihead X Project) provides an X Window System compatible environment where multiple displays connected to multiple machines are presented as a single unified screen. In DMX, a master node distributes X Window primitives on a tiled display.

All of these approaches assume that a single application occupies the entire wall at any given instant. Rendering of the content is often conducted directly on the graphics cards that are connected to the displays. This has the advantage that it enables low-latency generation and manipulation of the images. And the frame synchronization among display nodes can be achieved easily by implementing a networked barrier at the point that needs to be synchronized. A networked barrier works by having all display nodes send a message to a barrier server node, and wait while the barrier server counts the number of messages it has received. When all the messages have been received the barrier server broadcasts acknowledgements to every display node, which upon receipt unblocks the display nodes. The synchronization barrier can be implemented in several ways. In cluster computing, the Message Passing Interface (MPI) [2] is the de-facto communication mechanism among the nodes. MPI supports a barrier among all its communication nodes. This ensures that all progress is blocked until all the processes running on the nodes enter what is called a “barrier”. This approach is sufficient to ensure frame synchronization for single applications occupying an entire tiled display.

Total image synchronization across displays in a tiled-display wall is best achieved through a combination of hardware and software solutions and is required for the display of stereoscopic images [24]. In terms of hardware, the synchronization of vertical refresh across multiple graphics cards can be achieved using specialized genlock hardware that is built into advanced graphics cards such as those found in Nvidia’s Quadro series [21-23]. Alternatively, Jeremie et al. [17] presented a cost-effective approach using custom hardware and parallel port for distributing vertical refresh synchronization signals. This can be used with any graphics hardware. Combined with software synchronization methods presented in many papers, these can provide cost-effective solution for total frame synchronization between display nodes.

Chen et al. [7] discussed three different communication methods for cluster-based high-resolution display system in their paper. These are synchronized execution model where all render nodes have the same copy of the application instance (e.g. CGLX), primitive distribution model where a client distributes graphics primitives to render servers (e.g. Chromium), and pixel distribution model where a client renders and transmits only pixels to display servers. A synchronization barrier at a certain program execution point (such as before graphics buffer swap) can be directly used to ensure frame synchronization in the synchronized execution model, such as in [4, 7, 12, 33, 35]. In the primitive distribution model, which can also be considered as a centralized model in that only a single node that has application instance distributes graphics primitives or pixels to server nodes that render and display, the frame synchronization can be achieved implicitly leaving small asynchronies between display nodes [25], or a synchronization barrier can be used explicitly such as in [11, 14, 15, 20] for tighter frame synchronization. SAGE can be categorized as *parallel* pixel distribution model since multiple clients (applications) send pixels to multiple display servers (display nodes). In the earlier generations of SAGE, we implemented a frame synchronization scheme to support multiple applications by using multiple synchronization barriers (a barrier per application). However, this per-application based synchronization scheme was unable to provide the tight synchronization tolerances expected by display

manufacturers such as Sharp. Our new approaches provide significant improvements.

3. Frame Synchronization Requirements of a Tiled Display Walls

To display a continuous image on a tiled display, all the tiles that constitute an application window need to be synchronously updated. This is especially important for interactive visualizations and animations.

There are three requirements for seamless frame synchronization on tiled displays:

1. Data synchronization: the application data to be displayed must be coherent i.e. the various display nodes must display parts of the same frame. For multiple applications, data synchronization must be achieved for each application being displayed.
2. Swap buffer synchronization: the display thread on each node should swap the contents of the graphics buffer synchronously in order for the various application windows to appear consistent on the display.
3. Synchronization of the vertical refresh cycles of the various displays (gen-lock): the physical refresh of monitors on each node should occur synchronously.

Perfect frame synchronization on tiled displays is achieved by satisfying all three requirements. In this paper we will focus on all but the third requirement which can normally be achieved through the use of specialized hardware.

In the case of dedicated tiled display walls that are limited to running only a single application at a time, data and swap buffer synchronization can be ensured easily with a single synchronization barrier. [4, 8, 12-15, 19, 20, 24, 26, 28, 29, 35] However, the problem we are attempting to solve is more challenging because tiled display walls can have an arbitrary number of different application windows in which frame updates occur at different rates. If a frame synchronization method for tiled display system that runs single application is applied, it becomes per-application based synchronization which is not scalable due to excessive synchronization messages over network generated for each frame from each application. Also with per-application-based synchronization, it is difficult to obtain swap buffer synchronization across display nodes because each application sends frames at different rates. This can cause the events (data and swap buffer synchronizations) of the same application to become partially ordered on a cluster, that can lead to unsynchronized display of frames. For multiple applications to be displayed seamlessly on a tiled display wall, total ordering of data and swap buffer synchronization across all applications is required. The total ordering of events in a distributed system is described in detail in [18].

In Section 5, we will propose two scalable frame algorithms that ensure total ordering of data synchronization of all applications and swap buffer synchronization between display nodes with minimal impact on applications' frame rate and latency.

4. Scalable Adaptive Graphics Environment (SAGE)

SAGE is a cross-platform middleware system for driving ultra-resolution tiled displays. Unlike other approaches, such as Chromium, SAGE delegates the rendering of graphics to remotely located compute clusters, and relies on the use of high speed networks to stream the pixels of the visualization to the displays. This "thin-client" model has the advantage that large cluster farms or supercomputers can be brought to bear to render

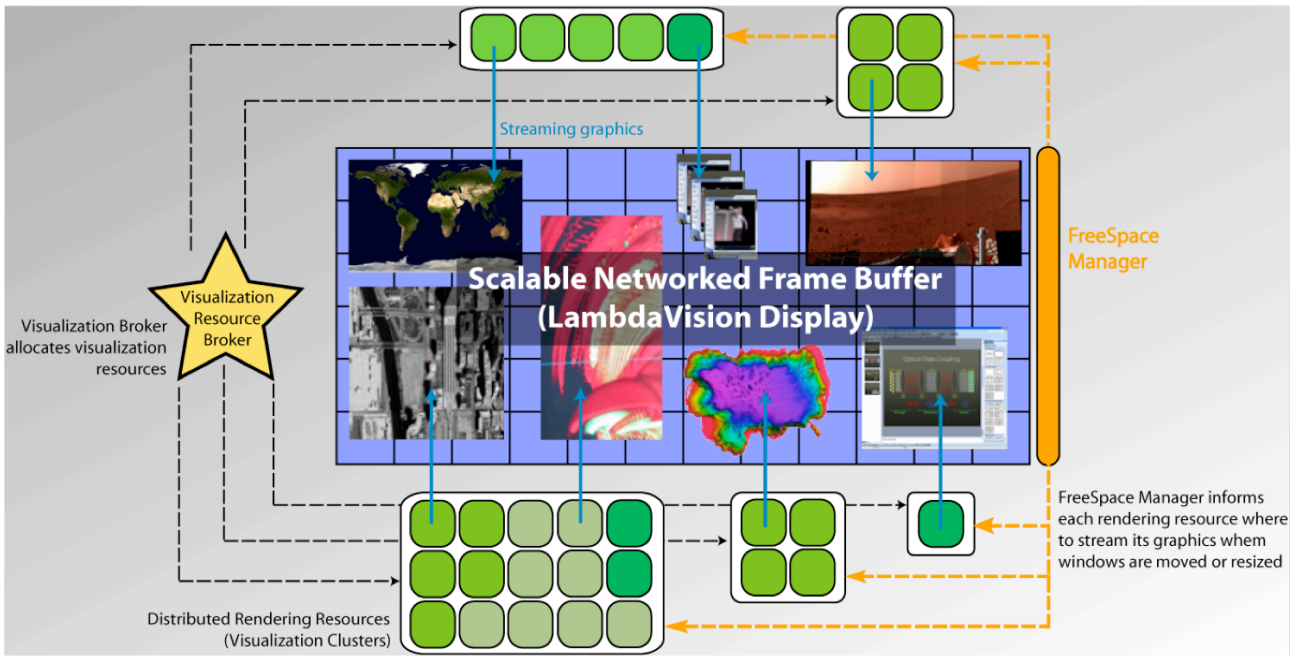

Figure 2. In SAGE, while a compute cluster drives the individual displays it merely acts as a light-weight client which receives pixels from remote rendering resources such as visualization clusters or supercomputers.

datasets that may be too large to fit on an individual graphics card [34]. In SAGE, a single window displayed on a wall may be driven by any number of display nodes, and multiple windows can be displayed on the wall simultaneously. As windows are moved from one portion of the wall to another, SAGE seamlessly reroutes the pixels to a different set of computers driving the display tiles so that handling of the windows on the display is totally transparent from the application. The SAGE model is shown in Figure 2.

4.1 Architecture

In SAGE, each application gives its rendered pixels to the SAGE Application Interface Library (SAIL) that streams them to the appropriate display nodes depending on the current position and size of the window on the display. Each display node can receive and display multiple pixel streams independently to allow multiple applications to be shown concurrently on the tiled display. The Free Space Manager (FSManager) is the main component of SAGE that keeps track of the current display parameters and the arrangement of the application pixels on the display. Based on the requested arrangement, the FSManager directs SAIL to distribute an application's pixels to the appropriate display nodes. The applications can be dynamically moved and resized with the help of the UI client. SAGE consists of following main components:

- SAGE Application Interface Library (SAIL) enables an application to stream pixel data onto the display wall.
- The application pixel streams are received by the SAGE Application Receiver threads (APP).
- Each node has a Node Display Manager (NDM) responsible displaying the contents of all applications on the display.
- Using the SAGE UI, a user can launch, resize, and move application windows on the tiled-display.

The Free Space Manager (FSManager) co-ordinates the various components. An example of SAGE session that runs on four display nodes and displays two applications is depicted in Figure 3.

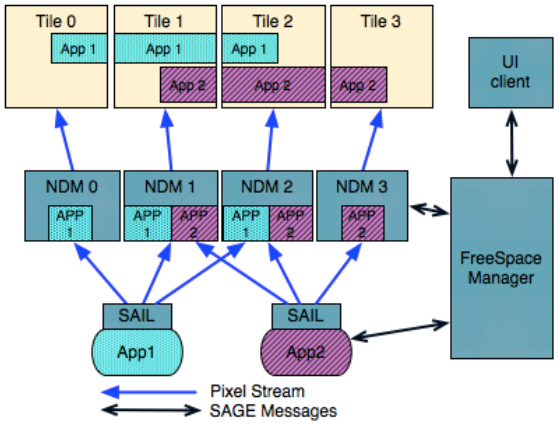

Figure 3. The SAGE Components. It shows an example of four display nodes running two applications, App1 and App2 each distributed on tile 0, 1, and 2 and tile 1, 2, and 3 respectively.

4.2 SAGE's Frame Synchronization Algorithm

In this section we will discuss the limitations of SAGE's former frame synchronization method. We employed a dynamic networked barrier per application for data synchronization. By handling multiple barriers at a time, we achieved data synchronization in SAGE [16].

A synchronization group (SyncGroup) which consists of a set of display nodes that shows an application's image is maintained for each application. Members (display nodes) in the group can be dynamically changed as a user moves or resizes an application window. And a data synchronization manager thread, which ensures synchronized frame update of display nodes in the SyncGroup, is created for each synchronization group. A dynamic SyncGroup and data synchronization manager thread pair implements a dynamic barrier for each application. An example of data synchronization manager and synchronization group pairs in SAGE is shown in Figure 4.

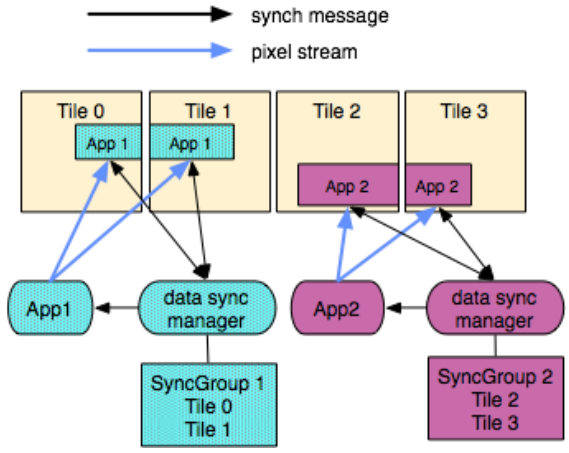

Figure 4. An example of four display nodes (tiles) displaying two applications. There is a data sync manager and SyncGroup pair for each application.

Table 1. The number of messages that need to be exchanged in SAGE in a single round with its former synchronization algorithm to display application data on the tiled-display.

	Number of Messages
Application Frame Updates	***M * N*** (Each application on a node sends a message to its sync master.) For M applications running on a display driven by N nodes, we have the worst case of ***M*N*** messages for frame updates.
Data Sync Messages	The worst case of ***M*N messages*** from data synch manager
Total messages per round	***(M*N) + (M*N) = 2*M*N***

Although this can achieve the data synchronization of multiple applications, this method requires excessive messages because it has a separate data synchronization manager for each application. Table 1 shows the worst-case message complexity when a tiled display consists of N display nodes displaying M applications. We also did not implement swap buffer synchronization explicitly. Since graphics swap buffer at each node can be performed as soon as data synchronization is finished, swap buffer synchronization can be achieved implicitly. Although this implicit swap buffer synchronization can be enough for supporting single application at a time [25], it may incur non-trivial synchronization jitter when it handles multiple applications because of the difference in CPU load and scheduling across display nodes. As the number of applications increases, the variance of the swap buffer completion time across all display nodes increases. Figure 5 illustrates implicit swap buffer synchronization that may result in incoherent frame display.

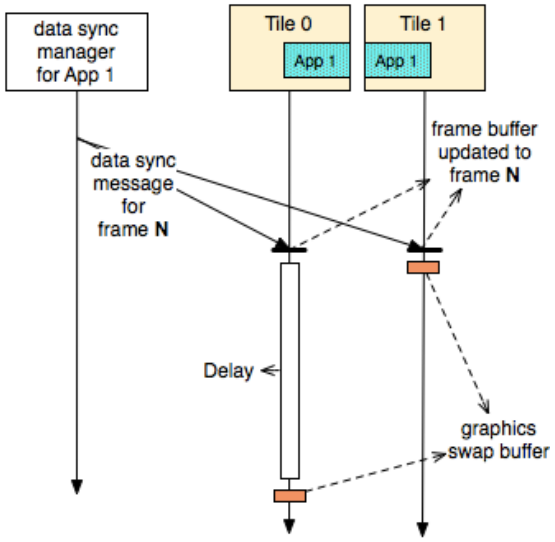

Figure 5. The effect of lacking swap buffer synchronization. The tile node 1 executed swap buffer right after updating to frame N, but the tile node 0 did not due to its CPU load and scheduling. This uncertain delay may increase as the number of applications on the tile 0 increases. This incurs frame synchronization jitter.

5. The Proposed Algorithms

In our approach, data synchronization is achieved by a single global synchronization master instead of a separate data synchronization manager for each application. We present two approaches in this paper – the Two-Phase algorithm and the One-Phase algorithm.

In both algorithms, the global synchronization manager provides data synchronization. Whereas the two-phase algorithm achieves swap buffer synchronization with a network barrier after the data synchronization phase, the one-phase algorithm uses NTP synchronized clocks on each node. The consequence of this is that the Two-Phase algorithm is more generally applicable to a variety of high-resolution tiled display systems, whereas the One-Phase approach yields higher synchronization accuracy but has a limited range of target systems since it requires the support for NTP and is more CPU-intensive.

5.1 Two-Phase Algorithm

This approach consists of two distinct phases;

1. the first phase achieves data synchronization for all applications, and
2. the second phase synchronizes the swap-buffer events of the all display node.

Figure 6 depicts the Two-Phase synchronization algorithm. In this case, we have a single global synchronization master (*SYNC MASTER*). Upon receiving a new frame at a display node, a corresponding application receiver on a display node sends a message with the new frame number and the node ID of the application receiver to the *SYNC MASTER*. The *SYNC MASTER* has an interval timer that runs at a periodic rate called **Sync Refresh Rate (SRR)**. The SRR must be a rate greater than the highest frame rate of all the applications in order to refresh all the applications at their desired rate. When the timer expires, the *SYNC MASTER* computes the highest common frame number for each application on all the nodes. After computing the highest common frame for each application, the *SYNC MASTER* sends a broadcast message to the NDM on each node. This message contains a list of the highest common frame number for each application. The NDM on each node uses this list in order to display the appropriate frame for each application. This concludes the first phase which achieves data synchronization.

Table 2. The number of messages that need to be exchanged in SAGE in a single round with the Two-Phase algorithm to display application frames on the Tiled-Display.

	Number of Messages
Application Frame Updates	***M * N*** (Each application on a node sends a message to the *SYNC MASTER*.) For M applications running on a display driven by N nodes, we have ***M*N*** messages for frame updates.
Phase 1: Data Sync Messages	***N*** (The *SYNC MASTER* sends a message to an NDM on all the ***N*** nodes.)
Phase 2: Swap Buffer Sync	***2N***
2a. Barrier msg from each NDM to the Barrier Master	N
2b. Broadcast msg from the Master to all NDM's	N
Total Messages per round	***(M*N) + N + 2N = (M+3)* N***

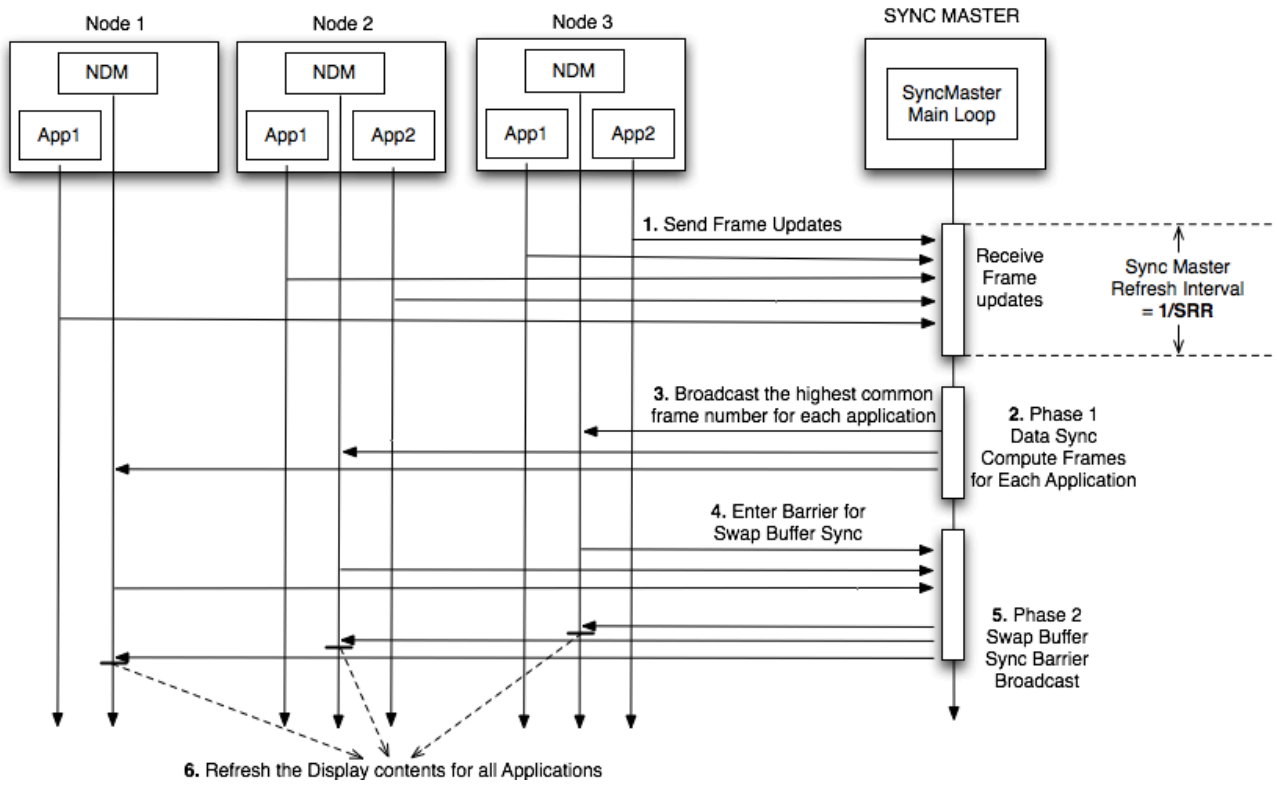

Figure 6. The Two-Phase synchronization algorithm to achieve seamless display of multiple applications on a tiled-display wall. The first phase ensures data synchronization, and the second phase ensures swap buffer synchronization using networked barrier.

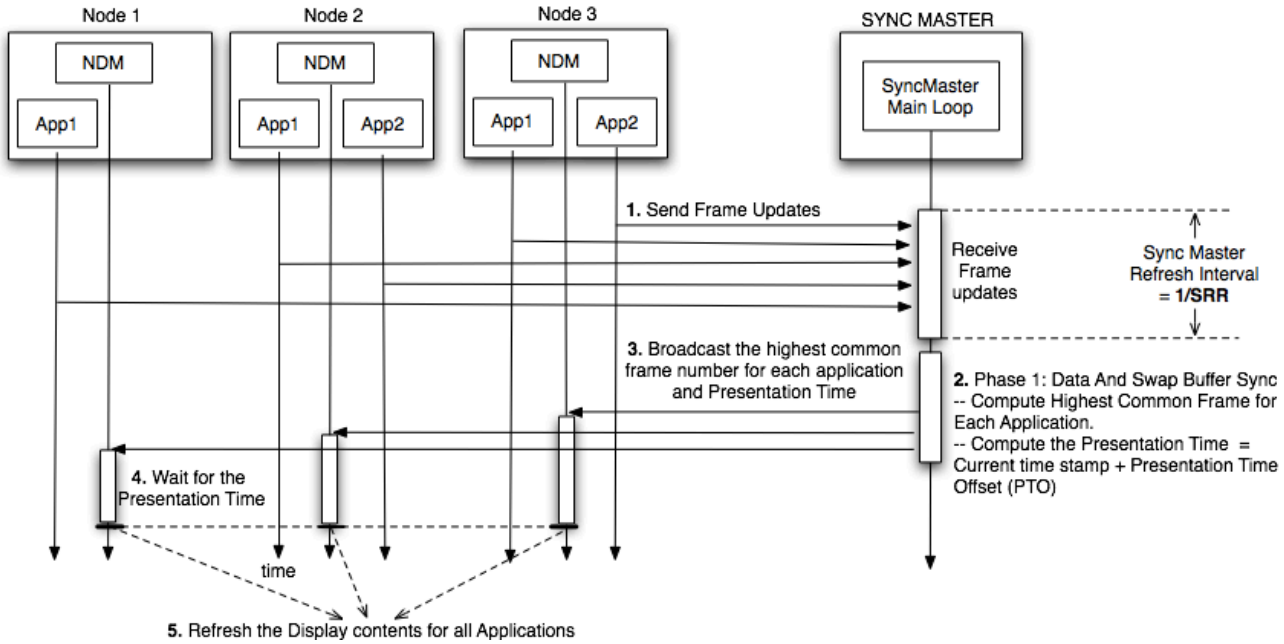

Figure 7. The Ones-Phase synchronization algorithm to achieve seamless display of multiple applications on a tiled-display wall. The first phase ensures data synchronization, and the synchronized swap buffer is ensured by making each node wait until Presentation Time instead of using centralized networked barrier at the SYNC MASTER.

Upon finishing the first phase, the NDM on each node enters the second phase in order to synchronize their swap buffer event. This is achieved by placing a networked barrier right after frame buffer drawing and just before frame buffer swap in the NDM. This enables swap buffer synchronization with each NDM displaying multiple applications on the tiled-display wall. Table 2 depicts the number of messages needed per round with the Two-Phase algorithm where ***M*** is the number of applications and ***N*** is the number of nodes driving the display wall. The Two-Phase method uses ***(M-3) * N*** messages less than the former SAGE frame synchronization algorithm per round.

The key differences between the Two-Phase and the former SAGE frame synchronization algorithm are:

1. The prior version of SAGE has a data synchronization manager for each application while the Two-Phase algorithm has a single data synchronization manager responsible for all the applications. This drastically reduces the number of messages needed to reach data synchronization per round.
2. The prior approach does not achieve swap buffer synchronization explicitly, whereas the Two-Phase algorithm uses a networked barrier to ensure swap buffer synchronization.

5.2 One-Phase Algorithm

In the One-Phase approach, we achieve both data and swap buffer synchronization in a single phase. We avoid the 2nd phase in the Two-Phase approach by synchronizing the clocks of the nodes driving the tiled-display. The Network Time Protocol (NTP), a common component in most major operating systems including Linux, helps synchronize the clocks on the cluster nodes.

Table 3. The number of messages that need to be exchanged in SAGE in a single round with the One-Phase algorithm to display application data on the Tiled-Display.

	Number of Messages
Application Frame Updates	***M * N*** (Each application on a node sends a message to the *SYNC MASTER*.) For M applications running on a display driven by N nodes, we have ***M*N*** messages for frame updates.
Phase 1: Data Sync Messages	***N*** (The *SYNC MASTER* sends a message to an NDM on all the ***N*** nodes. The Presentation Time is embedded in the message.)
Total Messages per round	***(M*N) + N = (M+1) * N***

The data synchronization procedure is identical to the 1st phase of the Two-Phase approach. A new term introduced in the algorithm is the **Presentation Time (PT)** that informs each NDM of the time when they should swap their buffer contents. After computing the highest common frame for each application, the *SYNC MASTER* computes PT by adding a **Presentation Time Offset (PTO)** to the current time. The *SYNC MASTER* sends a broadcast message to each NDM. This message contains the Presentation Time and a list of the highest common frame number for each application. Each NDM waits till the Presentation time and then displays the appropriate frame for each application according to the highest common frame number list. This procedure achieves both data and swap buffer synchronization.

The Presentation Time Offset (PTO) depends on a number of factors including the computational load on each node, the message delivery time, and the maximum frame rate of all applications. PTO can be either fixed to a constant large value or computed dynamically by the *SYNC MASTER* based on periodic feedback from the various clients. In our prototype, we chose fixed value empirically.

The trade-off of the One-Phase algorithm against Two-Phase algorithm is lower synchronization jitter in return for higher CPU usage due to the implementation limit. However, due to unpredictable nature of user interaction, network status, and computational load at each node, it is hard to determine proper PTO for each frame (for each round) adaptively at the *SYNC MASTER*.

6. Experiments

In this section, we evaluate the efficacy of the Two-Phase and One-Phase approach and compare them with the prior SAGE synchronization algorithm. We evaluate how the two new algorithms scale with respect to the number of applications, number of nodes, and the frame rate of applications. In Figure 9 through 13, “old sage” refers to the prior version of SAGE that does not use the enhanced synchronization methods. In the case of the One-Phase approach, the wait period till the Presentation Time is implemented using high-resolution hardware counters.

The testbed consists of a 28-node cluster driving an 11x5 tiled display wall. The cluster nodes are each equipped with 64-bit dual processor 2.4Ghz AMD Opterons with 4GB RAM, Nvidia Quadro 3000 Graphics Card in an AGP slot, and a dedicated 1GigE network interface card (NIC). The cluster nodes run Linux kernel 2.6. The nodes are interconnected via a CISCO 3750 switch with 96Gbps bisection bandwidth. A software-based NTP server is run on the master node and a NTP client is run on each cluster node to synchronize the clocks. The NTP protocol has an accuracy of 100 microseconds.

For the test application, we use several 1K (approximately 1000x1000 pixel) animations stored on a high-end dual-processor dual-core AMD Opteron node which is equipped with 8GB RAM and is connected to the 28-node cluster over a 10gigabit network via a 10G Neterion NIC.

We use the difference in the swap buffer completion time among the various nodes as a metric to evaluate inter-tile frame synchronization. Figure 8 depicts the swap buffer completion time that is defined as the time difference between the earliest and the latest swap buffer completion times among all nodes for a particular refresh cycle. A large difference indicates the tiles are out-of-synch. In the case of video-playback, which is our main concern in this paper, the simultaneous frame transition across display nodes should occur in several millisecond [25].

Since the clocks are synchronized using NTP, the swap buffer completion time is measured by time-stamping the swap buffer completion events on each node.

We will show our experiments on the inter-tile swap buffer completion time differences for a single application in the Section 6.1 and for multiple applications in the Section 6.2. In Sections 6.3 and 6.4, we will show the effect of the synchronization method on the frame rates of a single and multiple applications. The scalability with respect to the number of display nodes is shown in the Section 6.5. In the Section 6.6, we compare the average CPU usage of the NDM of the Two-Phase and the One-Phase algorithms.

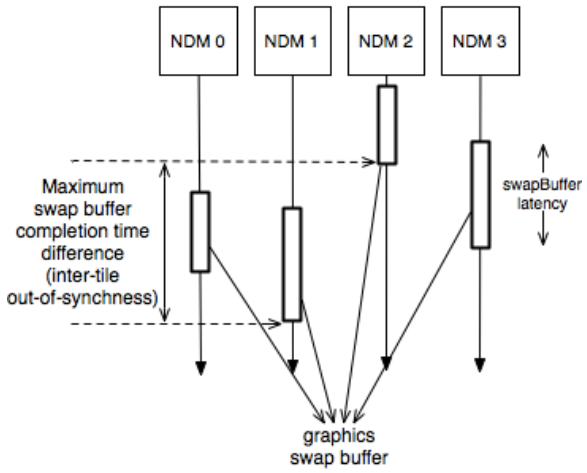

Figure 8. The method to compute the difference in the swap buffer completion time at each frame among four display nodes.

6.1 Inter-Tile Swap Buffer Completion Time Difference (Out-of-Sync Time) for a Single Application

A single test application was run on the entire display at 30 frames per second. The *SYNC MASTER* Refresh Rate (SRR) was set to 60Hz and the presentation time offset (PTO) for the One-Phase method was configured to 9 ms. Figure 9 depicts the swap buffer completion time variance observed for 1000 consecutive frames. As seen in the figure, the difference in the completion time of swap buffer among the various nodes in the prior approach is around 12-14 ms which caused the viewers to visually perceive tearing in the application image on the display wall. The Two-Phase algorithm achieves around 2 ms - A **6-fold** improvement over the prior approach. The One-Phase algorithm achieves around 0.15 ms - a **10-fold** improvement over the Two-Phase and up to **90-fold** improvement over the prior approach. The relatively high variance in "old sage" is due to lacking swap buffer synchronization (i.e. the second phase of the Two-Phase algorithm) in NDMs. This showed the need for swap buffer synchronization even for the case of single application. The swap buffer completion time variance in the Two-Phase algorithm was higher than that of the One-Phase algorithm because of the barrier latency in the second phase of the Two-Phase algorithm. However, both the Two-Phase and One-Phase algorithms achieve inter-tile frame synchronization and exhibit a visually seamless display across the wall.

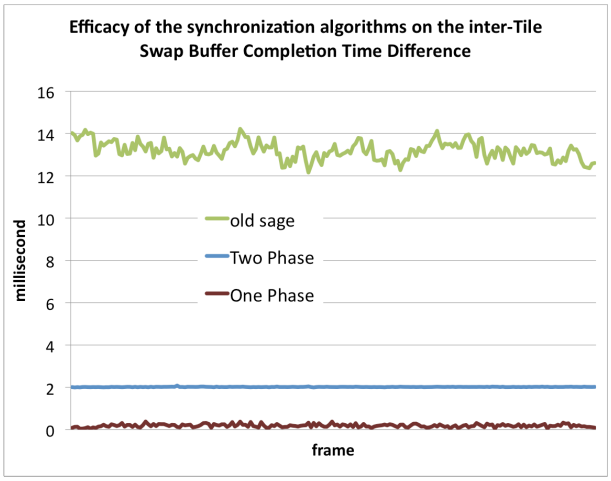

Figure 9. A comparison of the swap buffer completion time differences. The graph shows that the proposed synchronization algorithms achieve extremely low swap buffer variance in comparison to the prior frame synchronization method in SAGE - which in turn results in better inter-tile frame synchronization. The high variance in the prior method is mainly due to lacking swap buffer synchronization.

6.2 Inter-Tile Swap Buffer Completion Time Difference (Out-of-Sync Time) with Increasing Number of Applications

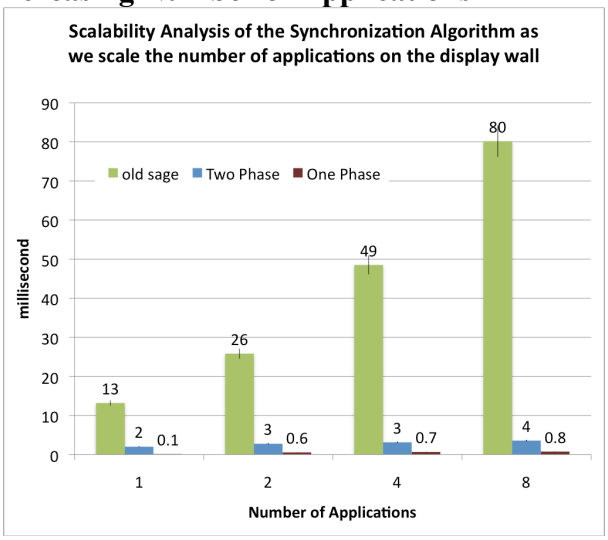

Figure 10. Average of maximum swap buffer completion time difference of multiple applications. The graph shows the algorithm scales as the number of applications is increased.

In this experiment, we increased the number of applications streaming to the display wall and evaluate the impact of the synchronization mechanisms on the inter-tile swap buffer

completion time. The *SYNC MASTER* Refresh Rate (SRR) was set to 60Hz and the Presentation Time Offset (PTO) for the One-Phase algorithm was fixed at 9 ms. The test application is a 1K animation remotely streamed at 30fps.

The results indicated that the prior approach, per-application synchronization, failed to sustain acceptable synchronization jitter with increasing number of applications, whereas both the Two-Phase and the One-Phase algorithms ensured tight frame synchronization despite of increasing number of applications. This was due to the single global synchronization master for all applications. Though minor increments of the swap buffer completion time difference were incurred by the increased system overhead on the display nodes as we increase the number of applications, this result still satisfied very tight frame synchronization tolerance.

6.3 Evaluation on an Application's Frame Rate

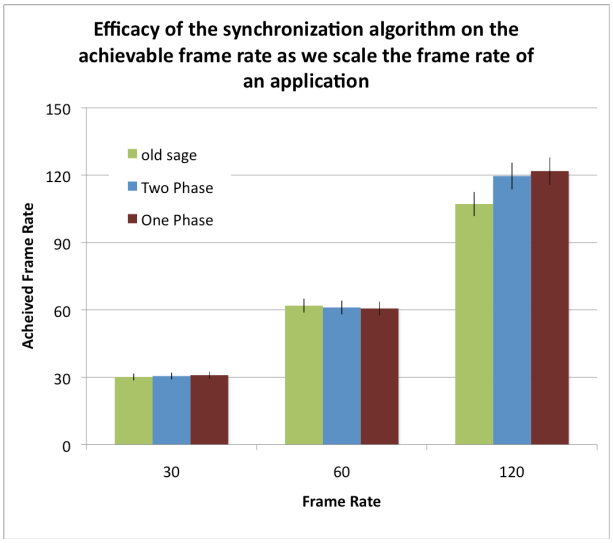

Figure 11. Comparison of the synchronization algorithms on the application frame rate as we scale the frame rate of a single application. The Two-Phase as well as the One-Phase algorithms scale with an application's frame rate.

3D stereoscopic tiled display walls including the Varrier [27] and StarCAVE [10] are used for interactive and immersive stereo visualizations. Such applications require support for a high frame rate up to 120 fps to achieve interactivity [6]. A frame synchronization scheme for these applications must be able to achieve tight synchronization with minimal impact on the application's frame rate. Though the current prototypes of these systems are designed for a single application, we expect the future extension for multiple applications. Thus, we evaluated the performance of the synchronization algorithms as we scaled the frame rate of a 1K animation displayed across the entire 28-node tiled display wall. In each case, the *SYNC MASTER* Refresh Interval was set 10Hz higher than the application rate and the Presentation Time Offset (PTO) for the One-Phase was set to 6ms. Figure 11 depicts the effect of the synchronization algorithms on the frame rate of the animation as we increased the target frame rate of the animation from 30fps to 120fps. From the figure, we observed that the two new algorithms were able to sustain the target frame rate with minimal deviation. The prior algorithm sustained the target frame rate till 60fps but failed to sustain it at 120fps.

6.4 Aggregate Frame Rate with Increasing Number of Applications

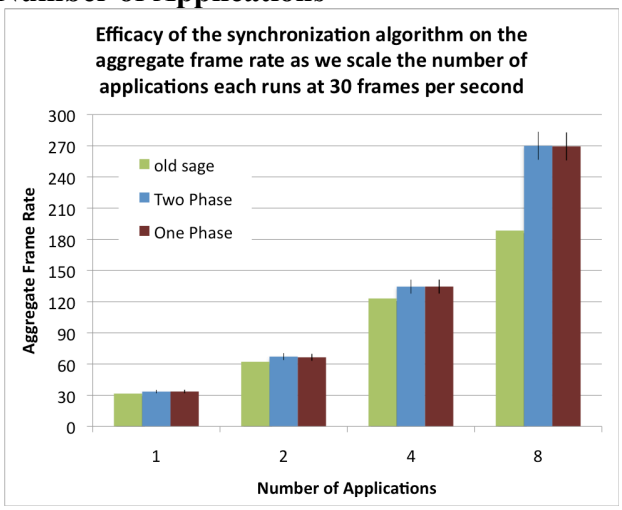

Figure 12. Comparison of the synchronization algorithms on the aggregate application frame rate as we increased the number of applications each runs at 30 frames per second. The Two-Phase and One-Phase methods scaled with the number of applications while the old SAGE synchronization algorithm showed its limited scalability.

In addition to being able to sustain high frame rates, a good synchronization mechanism should be able to sustain the frame rates as increasing number of applications are launched on a tiled display wall. In this experiment, we increase the number of applications streaming to the display wall and evaluate the impact of the synchronization mechanisms on the aggregate achievable frame rate. Again, each application streams a 1K animation at 30fps. From Figure 12, we observed that the synchronization mechanisms of the previous version SAGE ("old sage") was only able to sustain the frame rate for up to four applications and showed a **25%** drop for eight applications. As indicated in Section 4, this is due to the fact that the per-application based data synchronization mechanism requires excessive synchronization messaging. In contrast, both the Two-Phase and the One-Phase algorithms are able to scale as the number of applications increases.

6.5 Scalability Analysis with Increasing Number of Nodes

Another key requirement of a good synchronization mechanism is the ability to scale with the number of display nodes. Figure 13 depicts the inter-tile swap buffer completion time differences as we increased the number of display nodes and the associated tiled-display size. In the case of the Two-Phase algorithm, we observed minor increments in the inter-tile swap buffer completion time difference as we scaled the number of nodes. This was primarily an effect of the networked swap buffer

barrier employed in the 2nd phase: as the number of nodes increased the time to broadcast all the messages increased and incurred additional lag. However, the 2 ms difference was too short for a user to notice any asynchrony between tiles. The One-Phase algorithm achieved the tightest synchronization (~0.02 milliseconds), primarily due to the synchronized clocks and use of high-resolution hardware timers for displaying the frame as close as possible to the presentation time. Again, the two new algorithms showed much more improved scalability than the prior approach in this test.

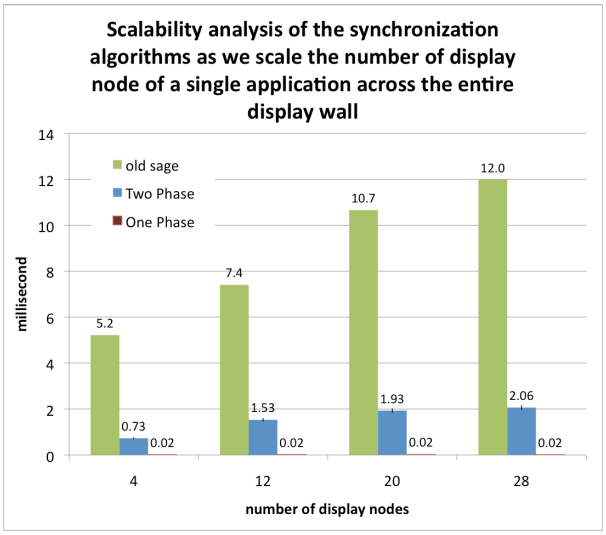

Figure 13. The graph shows scalability of the algorithms as the number of display nodes increases. The swap buffer completion time difference in the Two Phase algorithm slightly increases due to swap buffer synchronization phase using a networked barrier, whereas the effect of the display node increase is minimal in the One Phase algorithm.

6.6 Comparison of Average CPU Utilizations of the Two-Phase and the One-Phase Synchronization Algorithms

We discuss the differences in resource utilizations of the Two-Phase and the One-Phase algorithm in this section. In the One-Phase algorithm, each display node enters into busy waiting loop, waits until the presentation time, and then executes graphics swap buffer. This achieves a synchronized swap buffer across display nodes and eliminates 2***N*** network messages where ***N*** is the size of the cluster (total ***N*** number of display nodes). Thus, the One-Phase algorithm reduces message complexity at a cost of more CPU cycles than the Two-Phase algorithm due to the wait-loop. Therefore, when a tiled display is driven by a cluster of thin-client computers that do not have enough CPU resource for the wait-loop, or computationally intensive processes need to run on the tiled display cluster, the One-Phase algorithm should be avoided. In this experiment, we evaluated CPU usages of the NDM when a 1K animation was displayed on the display wall. The One-Phase algorithm used 20% of CPU time, whereas the Two-Phase algorithm used only 4%.

7. Discussion

The Presentation Time Offset (PTO) of the One-Phase algorithm has been set manually in the current implementation. PTO has to be carefully chosen otherwise it can fail swap buffer synchronization or cause unnecessary busy waiting. If it is set too small, a synchronization message can arrive at a display node behind the Presentation Time (PT). Or a node can complete the frame buffer drawing behind PT. The swap buffer synchronization fails in these cases. If it is set too long, then extra busy waiting can occur at a display node resulting in wasted CPU cycles. An example of small PTO is shown in Figure 15. The NDM3's graphics swap buffer is not synchronized because the synchronization message arrives at the NDM3 behind PT. To adaptively determine PTO for each frame at the *SYNC MASTER* can resolve the problems described above. However, to make PTO adaptive is challenging due to the uncertainty in the system load on the cluster.

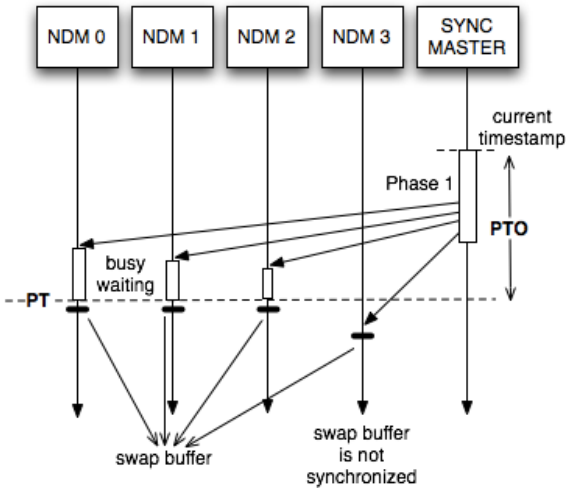

Figure 15. The effect of small PTO. The Presentation Time(PT) calculated from the Presentation Time Offset(PTO) was passed when the synchronization message is received at NDM3. Graphics swap buffer at NDM3 cannot be synchronized with swap buffer on other display nodes.

The two algorithms we have presented assume that frames are delivered reliably- i.e. the image frame data must not be lost or dropped in any of the display nodes. Therefore, the current prototypes are not applicable to the case where an application may use UDP for streaming- this is a subject of future investigation.

8. Conclusion

We presented the Two-Phase and the One-Phase algorithms to achieve a seamless display of multiple applications on a high-resolution tiled display wall driven by a cluster of computers. Whether one would choose to utilize the One-Phase versus Two-Phase algorithm depends on the desired synchronization accuracy and the availability of system resources. The Two-Phase algorithm has the advantage that it is more generic and can therefore be easily applied to most high–resolution tiled

display systems including the ones driven by networked thin clients. It provides high synchronization accuracy generally acceptable for interactive high-resolution visualization. The One-Phase algorithm provides superior synchronization characteristics due to its low degree of messaging complexity. However, the One-Phase algorithm requires support for NTP and sophisticated display thread scheduling (which is currently implemented via a busy-wait loop). Hence, if very tight synchronization is required and NTP and additional unused processing cores are available, one should opt to use the One-Phase algorithm. Both methods, however, will scale with respect to the number of applications, the frame rates of the applications, and the number of cluster nodes.

9. Acknowledgements

This work was supported in part by the Office of Advanced Scientific Computing Research, Office of Science U.S. Department of Energy, under Contract No. DE-AC02-06CH11357. This project was also funded in part by National Science Foundation Award: OCI 0943559.

10. References

[1] Cross-Platform Cluster Graphics Library (CGLX), http://vis.ucsd.edu/~cglx/

[2] Message Passing Interface, http://www.mpi-forum.org/

[3] Xdmx, http://dmx.sourceforge.net

[4] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin, "Net Juggler: running VR Juggler with multiple displays on a commodity component cluster," In *Virtual Reality, 2002. Proceedings. IEEE*, vol., no., pp.273-274, 2002.

[5] R. Ball, and C. North: "Analysis of User Behavior on High-Resolution Tiled Displays," *Human-Computer Interaction, INTERACT 2005*, pp.350-363, 2005.

[6] C.-N. Carolina, J. S. Daniel, and A. D. Thomas: "Surround-screen projection-based virtual reality: the design and implementation of the CAVE," *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, 1993.

[7] H. Chen, Y. Chen, A. Finkelstein, T. Funkhouser, K. Li, Z. Liu, R. Samanta, and G. Wallace: "Data distribution strategies for high-resolution displays", *Computers & Graphics*, vol.25, no.5, pp.811-818, 2001.

[8] J.-D. Choi, K.-J. Byun, B.-T. Jang, and C.-J. Hwang: "A synchronization method for real time surround display using clustered systems," *Proceedings of the tenth ACM international conference on Multimedia*, Juan-les-Pins, France, pp.259-262, 2002.

[9] M. Czerwinski, G. Smith, T. Regan, B. Meyers, G. Robertson, and G. Starkweather, "Toward characterizing the productivity benefits of very large displays," In *Human-Computer Interaction, INTERACT 2003*, vol., no., pp.9-16, 2003.

[10] T. A. DeFanti, G. Dawe, D. J. Sandin, J. P. Schulze, P. Otto, J. Girado, F. Kuester, L. Smarr, and R. Rao: "The StarCAVE, a third-generation CAVE and virtual reality OptIPortal", *Future Generation Computer Systems*, vol.25, no.2, pp.169-178, 2009.

[11] S. Eilemann, M. Makhinya, and R. Pajarola: "Equalizer: A Scalable Parallel Rendering Framework", *Visualization and Computer Graphics, IEEE Transactions on*, vol.15, no.3, pp.436-452, 2009.

[12] J.-y. Huang, K. M. Wang, and K.-W. Hsu: "The frame synchronization mechanism for the multi-rendering surrounding display environment", *Displays*, vol.25, no.2-3, pp.89-98, 2004.

[13] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan: "Distributed rendering for scalable displays," *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, Dallas, Texas, United States, 2000.

[14] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan: "WireGL: a scalable graphics system for clusters," *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001.

[15] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski: "Chromium: a stream-processing framework for interactive rendering on clusters," *ACM SIGGRAPH ASIA 2008 courses*, Singapore, 2008.

[16] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, and J. Leigh: "High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment," *Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference*, pp.24-24, 2006.

[17] A. Jeremie, G. Valerie, L. Guy, M. Emmanuel, and R. Bruno: "SoftGenLock: active stereo and genlock for PC cluster," *Proceedings of the workshop on Virtual environments 2003*, Zurich, Switzerland, pp.255-260, 2003.

[18] L. Lamport: "Time, clocks, and the ordering of events in a distributed system", *Communications of the ACM*, vol.21, no.7, pp.558-565, 1978.

[19] R. B. M. Bues, S. Stegmaier, U. Häfner, H. Hoffmann, F. Haselberger, "Towards a Scalable High Performance Application Platform for Immersive Virtual Environements," In *Proceedings of Immersive Projection Technology and Virtual Environments*, vol., no., pp.165-174, 2001.

[20] Nirnimesh, P. Harish, and P. J. Narayanan: "Garuda: A Scalable Tiled Display Wall Using Commodity PCs", *Visualization and Computer Graphics, IEEE Transactions on*, vol.13, no.5, pp.864-877, 2007.

[21] Genlock, http://www.nvidia.com/object/IO_10793.html

[22] Nvidia Quadro G-Sync, http://www.nvidia.com/page/quadrofx_gsync.html

[23] Nvidia, Quadro FX 3000G Solutions for Advanced Visualization. Technical Report. NVIDIA Corporation, 2003.

[24] B. Raffin, L. Soares, N. Tao, R. Ball, G. S. Schmidt, M. A. Livingston, O. G. Staadt, and R. May, "PC Clusters for Virtual Reality," In *Virtual Reality Conference*, vol., no., pp.215-222, 2006.

[25] A. Rrustemi. 'Computing Surface - a platform for scalable interactive displays'. Doctoral Thesis, University of Cambridge, 2008.

[26] S. Rudrajit, Z. Jiannan, F. Thomas, L. Kai, and S. Jaswinder Pal: "Load balancing for multi-projector rendering systems," *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, Los Angeles, California, United States, 1999.

[27] D. J. Sandin, T. Margolis, J. Ge, J. Girado, T. Peterka, and T. A. DeFanti: "The Varrier™ autostereoscopic virtual

reality display," *ACM SIGGRAPH 2005 Papers*, Los Angeles, California, 2005.
[28] B. Schaeffer: "Networking and Management Frameworks for Cluster-Based Graphics," *Virtual Environment on a PC Cluster Workshop*, Protvino, Russia, 2002.
[29] D. R. Schikore, R. A. Fischer, R. Frank, R. Gaunt, J. Hobson, and B. Whitlock: "High-resolution multiprojector display walls", *Computer Graphics and Applications, IEEE*, vol.20, no.4, pp.38-44, 2000.
[30] L. Shupp, R. Ball, B. Yost, J. Booker, and C. North: "Evaluation of viewport size and curvature of large, high-resolution displays," *Proceedings of Graphics Interface 2006*, Quebec, Canada, 2006.
[31] L. Smarr, M. Brown, and C. de Laat: "Special section: OptIPlanet -- The OptIPuter global collaboratory", *Future Generation Computer Systems*, vol.25, no.2, pp.109-113, 2009.
[32] D. S. Tan, D. Gergle, P. Scupelli, and R. Pausch: "Physically large displays improve performance on spatial tasks", *ACM Trans. Comput.Hum. Interact*, vol.13, no.1, pp.71-99, 2006.
[33] G. Wallace, O. J. Anshus, P. Bi, H. Chen, Y. Chen, D. Clark, P. Cook, A. Finkelstein, T. Funkhouser, G. Anoop, M. Hibbs, K. Li, Z. Liu, S. Rudrajit, S. Rahul, and O. Troyanskaya: "Tools and applications for large-scale display walls", *Computer Graphics and Applications, IEEE*, vol.25, no.4, pp.24-33, 2005.
[34] D. D. Winter, P. Simoens, L. Deboosere, F. D. Turck, J. Moreau, B. Dhoedt, and P. Demeester: "A hybrid thin-client protocol for multimedia streaming and interactive gaming applications," *Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, Newport, Rhode Island, 2006.
[35] C. Yuqun, C. Han, D. W. Clark, L. Zhiyan, G. Wallace, and L. Kai, "Software environments for cluster-based display systems," In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, vol., no., pp.202-210, 2001.